

PENERAPAN METODE *MERGE SORT* DAN *SEQUENTIAL SEARCH* PADA SISTEM PENGELOLAAN DATA PRAKTIKAN

Vista Mellyna Atsfi¹, Anton Prafanto^{*1}, Novianti Puspitasari¹

¹Program Studi Informatika, Fakultas Teknik, Universitas Mulawarman, Samarinda, Indonesia
Penulis korespondensi: Novianti Puspitasari (novipuspitasari@unmul.ac.id)

Pengelolaan data praktikan Fisika Dasar di Fakultas Teknik Universitas Mulawarman sebelumnya dilakukan menggunakan beberapa berkas spreadsheet terpisah sehingga pencarian, pengurutan, presensi, penilaian, dan rekapitulasi nilai menghadapi risiko duplikasi serta kesalahan data. Penelitian ini bertujuan mengembangkan sistem pengelolaan data praktikan berbasis web dengan menerapkan Merge Sort untuk pengurutan dan Sequential Search untuk pencarian data. Pengembangan sistem menggunakan metode Waterfall yang meliputi analisis kebutuhan, perancangan, implementasi, pengujian, dan evaluasi. Sistem dibangun menggunakan Laravel dan MySQL dengan 538 data praktikan tahun akademik 2025 serta data sintesis berukuran 100, 500, 1.000, dan 5.000 entri. Pengujian fungsional menggunakan Black Box Testing, sedangkan pengujian kinerja mengukur waktu eksekusi, penggunaan memori, jumlah iterasi, dan jumlah perbandingan. Hasil pengujian menunjukkan seluruh fitur utama berjalan sesuai rancangan dan kedua algoritma menghasilkan waktu eksekusi serta penggunaan memori lebih rendah dibandingkan MySQL.

Kata Kunci – Black Box Testing, Merge Sort, Praktikan, Praktikum, Sequential Search.

I. PENDAHULUAN

Praktikum Fisika Dasar di Fakultas Teknik Universitas Mulawarman melibatkan mahasiswa dari berbagai program studi sehingga data yang dikelola mencakup identitas praktikan, kelompok, sesi, presensi, materi, penilaian, dan nilai akhir. Pada kondisi sebelumnya, data praktikan, presensi, dan penilaian dikelola menggunakan beberapa file spreadsheet secara terpisah. Pada satu periode tahun 2025, jumlah praktikan mencapai 382 mahasiswa untuk Fisika Dasar 1 dan 156 mahasiswa untuk Fisika Dasar 2. Fragmentasi tersebut menyebabkan pengelola perlu melakukan pemeriksaan silang ketika data diperbarui, termasuk saat memeriksa duplikasi NIM, mencari data, menyusun urutan, dan melakukan rekapitulasi. Penggunaan *spreadsheet* juga tetap memiliki risiko kesalahan akibat formula, struktur, dan pengelolaan data yang kurang terkendali [1].

Sistem berbasis web dapat digunakan untuk memusatkan penyimpanan dan pengelolaan data akademik sehingga informasi lebih mudah diakses dan

diperbarui dari satu sumber data. Pengembangan sistem akademik berbasis web menggunakan Laravel dan MySQL telah diterapkan untuk pengelolaan data akademik, nilai, dan presensi [2]–[4]. Namun, sentralisasi data saja belum menjawab kebutuhan pengolahan pada lapisan aplikasi. Ketika data bertambah, pengguna tetap memerlukan mekanisme pengurutan dan pencarian untuk menampilkan data sesuai kebutuhan.

Merge Sort dipilih untuk proses pengurutan karena menggunakan pendekatan *divide and conquer* dan memiliki kompleksitas waktu $O(n \log n)$ yang relatif stabil pada berbagai susunan data [5]–[7]. Sequential Search digunakan untuk pencarian karena dapat memeriksa elemen secara berurutan tanpa mensyaratkan data telah terurut. Penerapan Sequential Search pada aplikasi berbasis web juga telah digunakan untuk pencarian dokumen dan data institusional [8]–[11]. Penelitian terkait menunjukkan bahwa algoritma *sorting* dan *searching* dapat diterapkan pada aplikasi web, tetapi sebagian kajian menguji algoritma secara terpisah dari sistem administrasi yang digunakan pengguna. Penelitian ini menerapkan Merge Sort dan Sequential Search secara bersamaan pada lapisan aplikasi Sistem Praktikum Fisika Dasar.

Perbedaan penelitian ini terletak pada penerapan kedua algoritma secara langsung terhadap data praktikan yang telah dimuat ke memori aplikasi, kemudian membandingkan karakteristik kinerjanya dengan mekanisme MySQL. MySQL ORDER BY digunakan sebagai *baseline* pengurutan, sedangkan WHERE/LIKE digunakan sebagai *baseline* pencarian. Perbandingan dilakukan berdasarkan waktu eksekusi dan penggunaan memori pada ukuran *dataset* yang sama. Pendekatan ini tidak dimaksudkan menggantikan MySQL, melainkan melihat karakteristik pengolahan data pada lapisan aplikasi dan basis data dalam lingkungan pengujian yang sama. Tujuan penelitian adalah mengembangkan sistem pengelolaan data praktikan berbasis web serta mengevaluasi kinerja Merge Sort dan Sequential Search dibandingkan *baseline* MySQL pada variasi jumlah data.

II. METODE PENELITIAN

Penelitian ini menggunakan model pengembangan Waterfall karena kebutuhan sistem telah dirumuskan melalui observasi dan wawancara. Tahapan penelitian meliputi identifikasi masalah, pengumpulan data, perancangan, implementasi, pengujian, dan evaluasi.

Observasi dilakukan terhadap proses pengelolaan Praktikum Fisika Dasar, sedangkan wawancara dilakukan dengan koordinator praktikum untuk memperoleh kebutuhan pengelolaan data, pencarian, pengurutan, presensi, dan penilaian.

Data penelitian terdiri atas 538 data praktikan tahun akademik 2025. Untuk evaluasi kinerja digunakan *dataset* sintesis berukuran 100, 500, 1.000, dan 5.000 entri agar setiap metode diuji pada kondisi yang sama. Merge Sort diuji pada empat skenario susunan data, yaitu sudah urut, duplikat, *random ascending*, dan *random descending*. Sequential Search diuji pada empat kondisi pencarian, yaitu kata kunci ditemukan, tidak ditemukan, kosong, dan menghasilkan data duplikat.

Tabel 1. Skenario Pengujian Kinerja

Metode	Skenario	Parameter
Merge Sort	Sudah urut, duplikat, <i>random ascending</i> , <i>random descending</i>	Waktu, memori, <i>comparison</i> , proses <i>merge</i>
Sequential Search	<i>Keyword</i> ada, tidak ada, kosong, duplikat	Waktu, memori, iterasi, <i>comparison</i>
MySQL ORDER BY	ORDER BY NIM ASC	Waktu, memori
MySQL WHERE/ LIKE	Duplikat, kosong, <i>keyword</i> ada, tidak ada	Waktu, memori

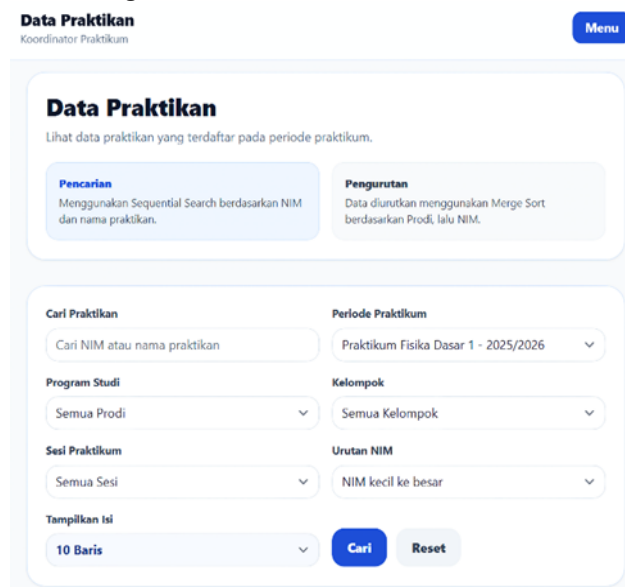
Pengukuran dilakukan pada sisi *server* menggunakan fungsi bawaan PHP. Waktu eksekusi dicatat dengan *microtime(true)*, sedangkan penggunaan memori diukur menggunakan *memory_get_peak_usage(true)*. Setiap skenario menjalankan 10 iterasi *warm-up* dan 50 iterasi utama. *Warm-up* tidak dimasukkan ke hasil akhir karena digunakan untuk mengurangi pengaruh inisialisasi dan kondisi awal sistem [12]. Hasil 50 iterasi utama disimpan pada tabel *log* pengujian untuk setiap metode. Data pengujian dianalisis menggunakan statistik deskriptif. Pembersihan *outlier* dilakukan menggunakan *Interquartile Range* (IQR) dengan batas bawah $Q1 - 1,5(IQR)$ dan batas atas $Q3 + 1,5(IQR)$. Dari 3.200 hasil pengujian utama, sebanyak 129 data dikeluarkan sebagai *outlier* sehingga diperoleh 3.071 data bersih.

Pengujian dilakukan pada komputer lokal (*localhost*) menggunakan sistem operasi Windows 11 64-bit, prosesor 12th Gen Intel(R) Core(TM) i5, RAM 12 GB, Solid State Drive (SSD), PHP 8.2.12, Laravel 12.57.0, dan Google Chrome. Pengukuran berfokus pada proses *server* sehingga tidak menghitung waktu tampilan (*rendering*) halaman dan latensi jaringan. Meskipun metode IQR dan *warm-up* telah diterapkan untuk mereduksi variasi data, potensi fluktuasi kinerja akibat penurunan kecepatan *clock* prosesor (*throttling*) maupun aktivitas proses latar belakang sistem operasi tetap dapat memengaruhi variasi waktu milidetik pada pengujian dengan iterasi tinggi.

III. HASIL DAN PEMBAHASAN

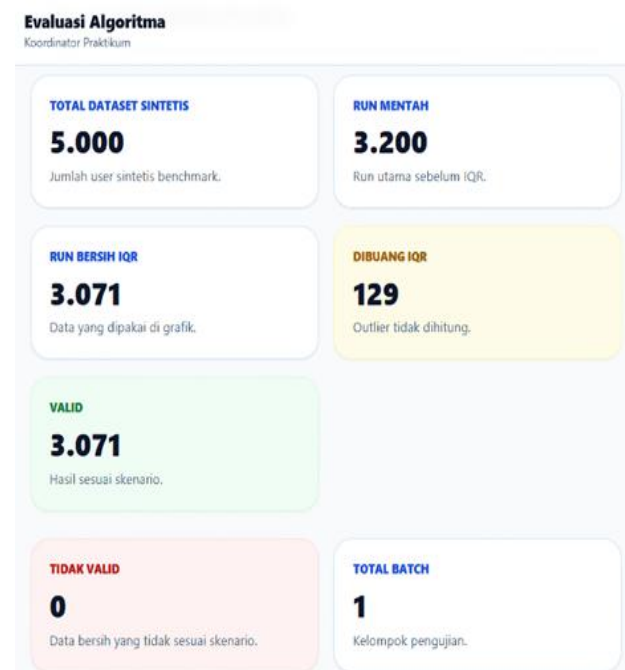
Sistem yang dikembangkan memusatkan pengelolaan data praktikan, asisten, periode praktikum, kelompok, materi, presensi, penilaian, dan nilai akhir dalam basis

data MySQL. Hak akses dibagi menjadi tiga peran, yaitu praktikan, asisten, dan koordinator praktikum. Praktikan dapat melihat data diri, kelompok, materi, jadwal, dan nilai. Asisten dapat mengelola presensi dan penilaian, sedangkan koordinator mengelola data periode, praktikan, kelompok, asisten, materi, kategori penilaian, serta evaluasi algoritma.



Gambar 1. Halaman Data Praktikan

Halaman data praktikan menyediakan pencarian menggunakan Sequential Search serta pengurutan menggunakan Merge Sort. Koordinator dapat memanfaatkan *filter* periode, program studi, kelompok, sesi, dan urutan NIM. Penggunaan kedua algoritma pada halaman ini menjadi bagian dari proses pengolahan ulang data yang telah dimuat dari basis data.



Gambar 2. Halaman Evaluasi Algoritma

Halaman evaluasi menampilkan hasil pengujian algoritma sehingga pengelola dapat melihat karakteristik performa berdasarkan ukuran dataset. Data benchmark digunakan secara terpisah dari data operasional agar proses evaluasi tidak mengubah data praktikan,

A. Hasil Pengujian Merge Sort

Pengujian Merge Sort dilakukan pada empat ukuran dataset dan empat skenario susunan data. Ringkasan rentang waktu dan penggunaan memori menunjukkan bahwa waktu eksekusi meningkat ketika jumlah data bertambah. Pada 100 data, waktu berada pada rentang 0,383057–0,592618 ms, sedangkan pada 5.000 data meningkat menjadi 27,659516–65,071449 ms. Memori meningkat dari sekitar 6,6328 KB menjadi 301,8672 KB. Data duplikat menghasilkan waktu tertinggi pada 5.000 data, sedangkan data yang sudah urut menghasilkan waktu terendah. Peningkatan waktu terutama berkaitan dengan bertambahnya elemen yang harus dibagi dan digabungkan selama proses pengurutan.

Tabel 2. Ringkasan Hasil Pengujian Merge Sort

Data	Rentang Waktu (ms)	Rentang Memori (KB)
100	0,383057–0,592618	≈6,6328
500	1,691477–3,763275	29,3047–29,4297
1.000	4,435458–6,549382	53,4141–53,5391
5.000	27,659516–65,071449	301,7422–301,8672

B. Hasil Pengujian Sequential Search

Sequential Search diuji pada kondisi *keyword* ada, tidak ada, kosong, dan duplikat. Waktu eksekusi meningkat ketika algoritma harus memeriksa lebih banyak elemen. Pada 100 data, rentang waktu berada pada 0,005438–0,064261 ms, sedangkan pada 5.000 data berada pada 0,009820–4,533239 ms. Waktu tertinggi terjadi pada skenario *keyword* ditemukan pada 5.000 data sebesar 4,533239 ms. Skenario *keyword* kosong tetap sangat rendah karena sistem tidak menjalankan pemeriksaan terhadap seluruh elemen. Penggunaan memori relatif stabil pada 0,3672–0,5000 KB karena algoritma hanya membutuhkan kata kunci, penghitung operasi, dan penampung hasil.

Tabel 3. Ringkasan Hasil Pengujian Sequential Search

Data	Rentang Waktu (ms)	Rentang Memori (KB)
100	0,005438–0,064261	0,3672–0,5000
500	0,007401–0,175257	0,3672–0,5000
1.000	0,007858–0,446066	0,3672–0,5000
5.000	0,009820–4,533239	0,3672–0,5000

C. Perbandingan dengan Baseline MySQL

MySQL ORDER BY dan WHERE/LIKE digunakan sebagai *baseline* untuk melihat perbedaan karakteristik pengolahan data pada lapisan basis data dan lapisan aplikasi. Hasil perbandingan menggunakan rata-rata seluruh skenario setelah pembersihan *outlier* menunjukkan bahwa Merge Sort memiliki waktu dan memori lebih rendah daripada ORDER BY pada seluruh ukuran *dataset* yang diuji. Pada 5.000 data, Merge Sort memiliki waktu rata-rata 45,504172 ms dengan memori 301,8359 KB, sedangkan MySQL ORDER BY mencapai 80,137558 ms dan 3.863,9141 KB.

Tabel 4. Perbandingan Merge Sort dan MySQL ORDER BY

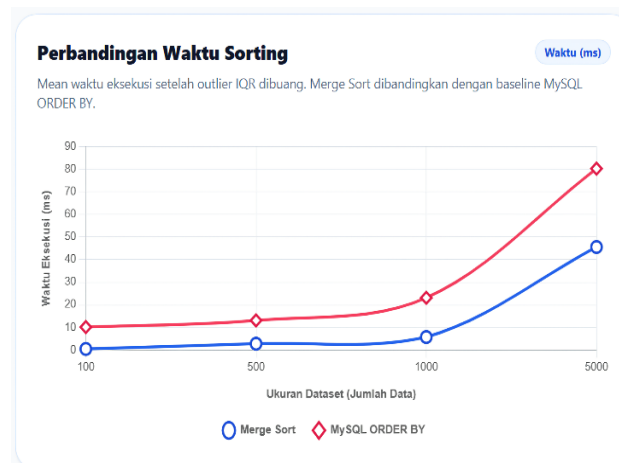
Data	Merge Sort (ms)	ORDER BY (ms)	Merge Sort (KB)	MySQL (KB)
100	0,449815	10,111763	6,6328	109,0000
500	2,758817	13,058431	29,3984	417,9063
1.000	5,709730	23,069350	53,5078	804,3516
5.000	45,504172	80,137558	301,8359	3.863,9141

Pada pencarian, Sequential Search juga memiliki rata-rata waktu lebih rendah daripada MySQL WHERE/LIKE. Pada 5.000 data, Sequential Search membutuhkan 2,534457 ms dengan memori sekitar 0,4648 KB, sedangkan WHERE/LIKE membutuhkan 55,276125 ms dengan memori 1.036,4063 KB. Perbedaan ini perlu dipahami berdasarkan lapisan pemrosesan: Sequential Search memeriksa data yang telah tersedia di memori aplikasi, sedangkan MySQL WHERE/LIKE menjalankan *query*, mengambil data dari basis data, dan mengembalikan hasil ke aplikasi.

Tabel 5. Perbandingan Sequential Search dan MySQL WHERE/LIKE

Data	Merge Sort (ms)	ORDER BY (ms)	Merge Sort (KB)	MySQL (KB)
100	0,043898	10,204491	0,4648	60,8770
500	0,126348	11,313327	0,4648	140,7324
1.000	0,294748	22,607393	0,4648	240,6406
5.000	2,534457	55,276125	0,4648	1.036,4063

Dengan demikian, hasil penelitian menunjukkan karakteristik performa pada konteks pengolahan ulang data yang sudah dimuat, bukan bahwa algoritma aplikasi selalu lebih baik untuk semua kebutuhan basis data.



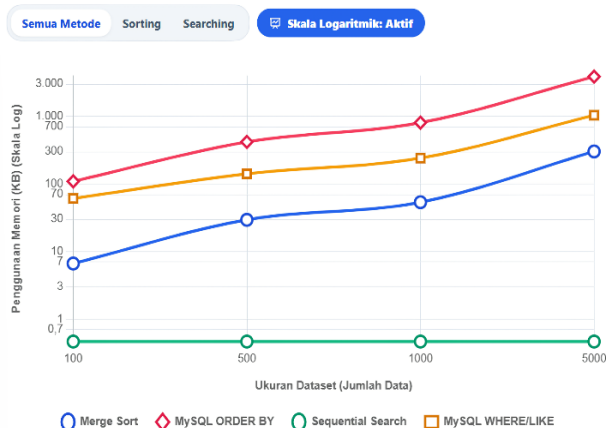
Gambar 3. Perbandingan Waktu Sorting

Tren pengujian menunjukkan peningkatan waktu pengurutan seiring bertambahnya ukuran dataset, dengan Merge Sort tetap lebih cepat daripada baseline ORDER BY. Perbedaan ini dipengaruhi oleh ruang lingkup pengukuran, yaitu Merge Sort memproses array dalam memori aplikasi, sedangkan baseline mencakup proses *query* basis data. Kedua metode menunjukkan peningkatan waktu pemrosesan, tetapi dengan karakteristik kenaikan yang berbeda.



Gambar 4. Perbandingan Waktu Searching

Pada proses pencarian, Sequential Search menunjukkan waktu yang lebih rendah dibandingkan WHERE/LIKE pada seluruh ukuran data. Meskipun demikian, MySQL tetap berperan sebagai sumber data utama. Hasil ini menunjukkan bahwa algoritma pada lapisan aplikasi dapat digunakan untuk pengolahan ulang data yang sudah dimuat, sedangkan *query* basis data tetap dibutuhkan untuk memperoleh data terbaru dan melakukan pencarian langsung pada penyimpanan.



Gambar 5. Perbandingan Penggunaan Memori

Merge Sort mengalami peningkatan penggunaan memori seiring bertambahnya ukuran *dataset*, tetapi nilainya tetap lebih rendah dibandingkan MySQL ORDER BY. Sequential Search menggunakan memori paling rendah dan relatif stabil pada seluruh ukuran *dataset*. Perbedaan tersebut menunjukkan bahwa kebutuhan memori dipengaruhi oleh mekanisme pengolahan data pada masing-masing metode.

D. Pembahasan

Hasil pengujian menunjukkan bahwa ukuran *dataset* menjadi salah satu faktor yang memengaruhi waktu eksekusi pada pengolahan data di lapisan aplikasi maupun basis data. Peningkatan jumlah data menyebabkan jumlah elemen yang harus diproses semakin besar sehingga waktu eksekusi cenderung meningkat. Pola tersebut terlihat pada Merge Sort dan Sequential Search, meskipun karakteristik kenaikannya berbeda sesuai mekanisme masing-masing algoritma. Hasil pengukuran juga menunjukkan bahwa perbedaan kondisi data dan pencarian dapat menghasilkan waktu yang berbeda pada ukuran *dataset* yang sama.

Pada Merge Sort, waktu eksekusi meningkat seiring pertambahan jumlah data. Pada 100 data, waktu rata-rata pada berbagai skenario berada pada rentang 0,383057–0,592618 ms, sedangkan pada 5.000 data meningkat menjadi 27,659516–65,071449 ms. Kenaikan tersebut berkaitan dengan bertambahnya elemen yang harus diproses melalui tahapan pembagian dan penggabungan. Kebutuhan memori Merge Sort meningkat mengikuti ukuran *dataset* karena proses penggabungan memerlukan ruang tambahan [7]. Karakteristik ini sesuai dengan kompleksitas waktu Merge Sort sebesar $O(n \log n)$, sehingga pertambahan jumlah elemen meningkatkan beban pemrosesan secara bertahap [5]–[7]. Hasil penelitian terdahulu juga menunjukkan bahwa Merge Sort dapat mempertahankan kinerja pengurutan pada ukuran data yang semakin besar [5], [6], [13].

Perbedaan kondisi awal data juga menghasilkan variasi waktu eksekusi Merge Sort. Pada 5.000 data, skenario data sudah urut menghasilkan waktu terendah sebesar 27,659516 ms, sedangkan data duplikat menghasilkan waktu tertinggi sebesar 65,071449 ms. *Random ascending* dan *random descending* masing-masing menghasilkan rata-rata 46,954765 ms dan 42,330955 ms. Perbedaan tersebut menunjukkan bahwa meskipun Merge Sort memiliki tahapan algoritmik yang sama, karakteristik isi data dapat memengaruhi waktu aktual yang tercatat selama pengujian. Kondisi data yang berbeda menyebabkan proses perbandingan dan penggabungan berlangsung dengan pola yang berbeda pada implementasi yang diuji.

Penggunaan memori Merge Sort juga meningkat sejalan dengan jumlah data. Pada 100 data, penggunaan memori berada pada sekitar 6,6328 KB, sedangkan pada 5.000 data mencapai sekitar 301,8672 KB. Peningkatan tersebut berkaitan dengan kebutuhan ruang tambahan selama proses penggabungan. Merge Sort menggunakan struktur tambahan untuk menampung elemen ketika proses merge dilakukan, sehingga kebutuhan memorinya turut dipengaruhi oleh ukuran *dataset* [7]. Pada ukuran *dataset* yang sama, perbedaan penggunaan memori antarskenario relatif kecil dibandingkan perbedaan yang disebabkan oleh jumlah data. Dengan demikian, jumlah elemen lebih terlihat sebagai faktor yang memengaruhi kebutuhan memori dibandingkan susunan awal data.

Sequential Search memperlihatkan karakteristik yang berbeda karena proses pencarian dilakukan dengan memeriksa data secara berurutan sampai kondisi pencarian terpenuhi atau seluruh data selesai diperiksa. Pada 100 data, waktu eksekusi berada pada rentang 0,005438–0,064261 ms, sedangkan pada 5.000 data rentangnya menjadi 0,009820–4,533239 ms. Waktu tertinggi pada 5.000 data terjadi pada kondisi *keyword* ditemukan, yaitu 4,533239 ms. Hasil tersebut menunjukkan bahwa pencarian dengan jumlah data yang lebih besar dapat membutuhkan pemeriksaan elemen yang lebih banyak. Karakteristik ini sesuai dengan sifat Sequential Search yang memiliki kompleksitas waktu linear $O(n)$ dan tidak mensyaratkan data dalam keadaan terurut [8]–[11].

Kondisi pencarian turut memengaruhi waktu yang dihasilkan Sequential Search. Skenario duplikat pada 5.000 data menghasilkan waktu rata-rata 3,391112 ms, sedangkan *keyword* tidak ditemukan menghasilkan 2,203659 ms. Pada kondisi *keyword* ditemukan, waktu dapat menjadi lebih tinggi karena proses pencarian dan pemeriksaan dilakukan terhadap sejumlah elemen hingga memperoleh hasil yang sesuai. Sebaliknya, kondisi *keyword* kosong hanya menghasilkan waktu 0,009820 ms pada 5.000 data. Nilai yang rendah tersebut bukan menunjukkan bahwa seluruh data dapat dicari dengan biaya yang sama, tetapi disebabkan implementasi sistem menangani *keyword* kosong melalui kondisi khusus sehingga pemeriksaan *keyword* tidak dilakukan terhadap seluruh elemen.

Penggunaan memori Sequential Search berada pada rentang 0,3672–0,5000 KB dan relatif stabil pada seluruh ukuran *dataset*. Kebutuhan tambahan tersebut berasal dari struktur yang digunakan selama pencarian, seperti

keyword, penghitung operasi, dan penampung hasil. Berbeda dengan Merge Sort yang membutuhkan ruang tambahan untuk proses penggabungan, Sequential Search tidak memerlukan struktur tambahan yang bertambah secara signifikan mengikuti jumlah elemen. Hasil ini sejalan dengan karakteristik pencarian linear yang menggunakan ruang tambahan relatif kecil ketika data sumber telah tersedia di memori aplikasi [8]–[11].

Perbandingan dengan MySQL menunjukkan adanya perbedaan karakteristik waktu dan penggunaan memori antara pengolahan data pada lapisan aplikasi dan basis data. Pada pengujian pengurutan, Merge Sort menghasilkan rata-rata waktu lebih rendah daripada MySQL ORDER BY pada seluruh ukuran *dataset*. Pada 100 data, rata-rata Merge Sort sebesar 0,449815 ms dibandingkan 10,111763 ms pada ORDER BY. Pada 5.000 data, rata-rata Merge Sort mencapai 45,504172 ms, sedangkan ORDER BY sebesar 80,137558 ms. Perbedaan tersebut perlu dipahami berdasarkan ruang lingkup pengukuran. Merge Sort menerima data yang telah tersedia di memori aplikasi, sedangkan pengukuran baseline MySQL mencakup proses *query* dan pengambilan hasil dari basis data ke aplikasi.

Perbedaan ruang lingkup tersebut juga terlihat pada penggunaan memori. Pada 5.000 data, Merge Sort menggunakan sekitar 301,8359 KB, sedangkan MySQL ORDER BY mencapai 3.863,9141 KB. Nilai tersebut tidak dapat diartikan bahwa Merge Sort menggantikan fungsi pengelolaan data MySQL. MySQL tetap menjalankan fungsi penyimpanan dan pengambilan data, sedangkan Merge Sort digunakan setelah data tersedia pada lapisan aplikasi untuk melakukan pengurutan ulang sesuai kebutuhan antarmuka sistem. Dengan demikian, hasil perbandingan lebih tepat dipahami sebagai karakteristik beban pemrosesan pada dua lapisan sistem yang memiliki ruang lingkup kerja berbeda.

Pada pencarian, Sequential Search juga menghasilkan waktu rata-rata yang lebih rendah daripada MySQL WHERE/LIKE pada *dataset* yang diuji. Pada 100 data, Sequential Search menghasilkan rata-rata 0,043898 ms, sedangkan WHERE/LIKE sebesar 10,204491 ms. Pada 5.000 data, waktu Sequential Search meningkat menjadi 2,534457 ms, sedangkan WHERE/LIKE mencapai 55,276125 ms. Penggunaan memori Sequential Search tetap sekitar 0,4648 KB, sementara penggunaan memori WHERE/LIKE meningkat dari sekitar 60,8770 KB pada 100 data menjadi 1.036,4063 KB pada 5.000 data.

Hasil penggunaan memori pada WHERE/LIKE juga dipengaruhi oleh jumlah data yang dikembalikan oleh *query*. Pada kondisi *keyword* kosong, seluruh data dapat dikembalikan sehingga penggunaan memori menjadi lebih besar. Kondisi *keyword* yang menghasilkan beberapa data duplikat juga membutuhkan ruang lebih besar untuk menampung hasil pencarian. Sebaliknya, ketika hasil pencarian hanya terdiri atas sedikit data, penggunaan memori yang tercatat lebih rendah. Kondisi tersebut menunjukkan bahwa beban pada *query* pencarian tidak hanya dipengaruhi oleh proses pencocokan, tetapi juga oleh jumlah data yang diambil dan dikirimkan kembali ke aplikasi. Mekanisme optimasi dan *indexing* pada MySQL juga dapat memengaruhi kinerja *query* pada kondisi tertentu [14], [15].

Berdasarkan analisis terhadap pengolahan data berbasis memori aplikasi (*in-memory*), terdapat rekomendasi praktis bagi pengembang sistem terkait ambang batas (*threshold*) jumlah data. Pemrosesan data di dalam memori mulai memerlukan perhatian khusus ketika ukuran *dataset* mencapai ambang batas sekitar 1.000 hingga 5.000 entri atau lebih pada skala produksi dengan akses banyak pengguna secara bersamaan (*concurrent users*). Pada kisaran volume data tersebut, eskalasi konsumsi memori puncak untuk struktur data dan operasi rekursif (seperti pemecahan *array* pada Merge Sort) mulai meningkat secara bertahap. Oleh karena itu, pengembang direkomendasikan untuk menerapkan teknik pembatasan halaman (*pagination*), *lazy loading*, atau memindahkan kembali mekanisme pengurutan dan pencarian ke tingkat basis data apabila jumlah data operasional melampaui batas ribuan entri, guna mengantisipasi risiko kehabisan alokasi memori (*memory exhaustion*) pada *server* produksi (*server* utama yang diakses langsung oleh pengguna).

Perbandingan menunjukkan bahwa Sequential Search dan Merge Sort sesuai untuk mengolah data yang telah dimuat ke memori aplikasi. MySQL tetap berfungsi sebagai sumber penyimpanan utama, sedangkan Merge Sort digunakan untuk mengurutkan data berdasarkan program studi dan NIM, serta Sequential Search untuk mencari data berdasarkan NIM atau nama. Dengan demikian, kedua algoritma berperan pada memori aplikasi tanpa menggantikan fungsi basis data.

Konsistensi hasil pengujian juga diperhatikan melalui pembersihan data menggunakan metode *Interquartile Range* (IQR). Dari 3.200 hasil pengujian utama, sebanyak 129 data dikategorikan sebagai *outlier* dan dikeluarkan dari analisis sehingga tersisa 3.071 data bersih. Pembersihan tersebut dilakukan untuk mengurangi pengaruh pengukuran yang menyimpang terhadap nilai rata-rata, median, dan standar deviasi. Penggunaan *warm-up* sebanyak 10 iterasi sebelum 50 iterasi utama juga diterapkan agar pengukuran tidak terlalu dipengaruhi kondisi awal eksekusi [12]. Dengan pendekatan tersebut, hasil analisis lebih berfokus pada pengukuran utama yang telah melalui proses penyaringan.

Nilai rata-rata, median, dan standar deviasi digunakan untuk menganalisis kecenderungan dan variasi waktu eksekusi. Rata-rata menunjukkan waktu eksekusi keseluruhan, median menunjukkan nilai tengah, sedangkan standar deviasi mengukur penyebaran data. Kedekatan rata-rata dan median mengindikasikan pengaruh nilai ekstrem yang relatif kecil, sementara standar deviasi yang tinggi menunjukkan variasi antarpengulangan yang lebih besar, yang dapat dipengaruhi oleh kondisi lingkungan pengujian, termasuk proses lain yang berjalan pada perangkat.

Secara fungsional, penerapan kedua algoritma mendukung kebutuhan pengelolaan data pada sistem praktikum. Pengujian Black Box Testing digunakan untuk menunjukkan bahwa fitur autentikasi, pembatasan hak akses, pengelolaan data praktikan, presensi, penilaian, pencarian, pengurutan, dan *filter* berjalan sesuai hasil yang diharapkan. Penggunaan Merge Sort dan Sequential Search tidak berdiri sendiri sebagai *fitur* pengujian, tetapi menjadi bagian dari proses pengolahan data pada halaman

praktikan. Data yang telah diperoleh dari basis data dapat diolah kembali berdasarkan kebutuhan pengguna tanpa mengubah data sumber pada penyimpanan.

IV. KESIMPULAN DAN SARAN

Penelitian ini menghasilkan sistem pengelolaan data praktikan Fisika Dasar berbasis web yang memusatkan data praktikan, asisten, periode, kelompok, materi, presensi, penilaian, dan nilai akhir. Hasil pengujian fungsional menggunakan Black Box Testing menunjukkan bahwa fitur utama dan hak akses berjalan sesuai rancangan.

Merge Sort berhasil diterapkan untuk pengurutan berdasarkan program studi dan NIM, dengan waktu 0,383057–0,592618 ms pada 100 data dan 27,659516–65,071449 ms pada 5.000 data, serta penggunaan memori meningkat dari sekitar 6,6328 KB menjadi 301,8672 KB. *Sequential Search* berhasil digunakan untuk pencarian berdasarkan NIM atau nama, dengan waktu 0,005438–0,064261 ms pada 100 data dan 0,009820–4,533239 ms pada 5.000 data, serta penggunaan memori relatif stabil pada 0,3672–0,5000 KB.

Secara keseluruhan, kedua algoritma menghasilkan waktu eksekusi dan penggunaan memori lebih rendah dibandingkan baseline MySQL pada lingkungan pengujian ketika data telah tersedia pada memori aplikasi. Temuan tersebut menunjukkan bahwa *Merge Sort* dan *Sequential Search* dapat mendukung pengolahan ulang data pada sistem praktikum berbasis web, sementara MySQL tetap digunakan sebagai penyimpanan utama dan sumber pengambilan data. Perbedaan hasil tersebut juga menunjukkan bahwa karakteristik kinerja algoritma perlu dipahami berdasarkan lokasi dan ruang lingkup proses pengolahan data yang diukur. Penelitian selanjutnya dapat memperluas ukuran *dataset*, melakukan pengujian pada *server* produksi dengan pengguna bersamaan, serta membandingkan kedua algoritma dengan algoritma *sorting* dan *searching* lainnya.

UCAPAN TERIMA KASIH

Penulis Vista Mellyna Atsfi mengucapkan terima kasih kepada orangtua, saudara, Bapak Anton Prafanto, S.Kom., M.T. dan Ibu Ir. Novianti Puspitasari, S.Kom., M.Eng. sebagai dosen pembimbing, teman-teman dan Tim Praktikum Fisika Dasar yang telah memberikan dukungan, arahan, dan bantuan selama proses pengembangan sistem dan penyusunan penelitian.

DAFTAR PUSTAKA

- [1] A. Gunawan, N. Wahyuni, R. Kiswindyatmoko, P. B. Katili, and D. R. Barleany, "Perancangan Sistem Informasi Pengelolaan Praktikum Berbasis Website," *Jurnal Teknik*, vol. 21, no. 1, pp. 17–31, Jun. 2023, doi: 10.37031/jt.v21i1.284.
- [2] E. Bagus Sadewa, Wijiyanto, and Nurohman, "Sistem Informasi Akademik Berbasis Web PadA SMK AL-ISLAM Surakarta," 2024. doi: <https://doi.org/10.24076/joism.2024v6i1.1647>.
- [3] R. J. Lontaan and A. R. Sinadia, "Design and Development of a Web-Based High School Information System," *COGITO Smart Journal*, vol. 10, no. 2, Dec. 2024, doi: <http://doi.org/10.31154/cogito.v10i2.858.593-606>.
- [4] L. Rahmawati and S. Sumarsono, "Desain Pengembangan Website dengan Arsitektur Model View Controller pada Framework Laravel," *Jurnal Teknologi Dan Sistem Informasi Bisnis*, vol. 6, no. 4, pp. 785–790, Oct. 2024, doi: 10.47233/jteksis.v6i4.1497.
- [5] M. Aditya, H. Zenitha, S. Wahyu, B. Novala, D. Muhammad, and A. N. Khudori, "JESICA: Journal of Enhance Studies in Informatics and Computer Applications Performance Analysis of Sorting Algorithm in Student Data Processing," vol. 2, no. 2, pp. 52–61, 2025.
- [6] O. Esau Taiwo, A. O. Christianah, A. N. Oluwatobi, K. A. Aderonke, and A. J. Kehinde, "COMPARATIVE STUDY of TWO DIVIDE and CONQUER SORTING ALGORITHMS: QUICKSORT and MERGESORT," in *Procedia Computer Science*, Elsevier B.V., 2020, pp. 2532–2540. doi: 10.1016/j.procs.2020.04.274.
- [7] Liu Peike, "An in-depth study of sorting algorithms," *Applied and Computational Engineering*, vol. 92, no. 1, pp. 187–195, Oct. 2024, doi: 10.54254/2755-2721/92/20241750.
- [8] Diana and Y. D. Putra, "Implementasi Algoritma Sequential Search Pada Aplikasi Pencarian E-Dokumen Berbasis Web," *CESS (Journal of Computing Engineering, System and Science)*, vol. 7, pp. 106–115, Jan. 2022, doi: <https://doi.org/10.24114/cess.v7i1.26114>.
- [9] A. Melati, D. Oktarina, Gusrianty, and W. Joni Kurniawan, "File Search in Employee Archives Using Sequential Search Algorithm," *IT Journal Research and Development*, vol. 7, no. 1, pp. 61–72, Aug. 2022, doi: 10.25299/ijrd.2022.9210.
- [10] Y. Reswan, Y. Darnita, A. R. Walad Mahfuzhi, and Elviani, "Implementation Of Sequential Searching Algorithm In Car Sales E-Commerce Web," *Jurnal Komputer, Informasi dan Teknologi*, vol. 4, no. 1, pp. 1–14, 2024, doi: 10.53697/jkomitek.v4i1.18.
- [11] Ridwan, Sunardi, and Alam, "Implementasi Algoritma Sequential Search dalam Pencarian Data Cuti pada Aplikasi Cuti Pegawai," *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, vol. 5, no. 1, pp. 198–206, Jan. 2025, doi: 10.57152/malcom.v5i1.1735.
- [12] L. Traini, V. Cortellessa, D. Di Pompeo, and M. Tucci, "Towards effective assessment of steady state performance in Java software: are we there yet?," *Empir. Softw. Eng.*, vol. 28, no. 1, Jan. 2023, doi: 10.1007/s10664-022-10247-x.
- [13] I. P. Pujiono, E. H. Rachmawanto, and N. A. S. Winarsih, "Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency Analysis," *Jurnal Manajemen Informatika (JAMIKA)*, vol. 15, no. 1, pp. 47–59, Apr. 2025, doi: 10.34010/jamika.v15i1.13230.
- [14] C. A. Györödi, D. V. Dumșe-Burescu, R. Györödi, D. R. Zmaranda, L. Bandici, and D. E. Popescu, "Performance impact of optimization methods on MySQL document-based and relational databases," *Applied Sciences (Switzerland)*, vol. 11, no. 15, Aug. 2021, doi: 10.3390/app11156794.
- [15] Y. Sahar, "Rancang Bangun Sistem Pelayanan Masyarakat Di Kantor Desa Sangtandung Berbasis Website," *Jurnal Riset Sistem Informasi*, vol. 1, pp. 27–36, Jul. 2024, doi: <https://doi.org/10.69714/9apd5w42>.